자료구조 Homework #8

충북대학교 컴퓨터공학과 2022040014 오재식

```c
/*
 *  doubly circular linked list
 *
 *  Data Structures
 *
 *  School of Computer Science
 *  at Chungbuk National University
 *
 */



#include<stdio.h>
#include<stdlib.h>
/* 필요한 헤더파일 추가 */

typedef struct Node {
    int key;
    struct Node* llink;
    struct Node* rlink;
} listNode;

/* 함수 리스트 */
int initialize(listNode** h);
int freeList(listNode* h);
int insertLast(listNode* h, int key);
int deleteLast(listNode* h);
int insertFirst(listNode* h, int key);
int deleteFirst(listNode* h);
int invertList(listNode* h);

int insertNode(listNode* h, int key);
int deleteNode(listNode* h, int key);

void printList(listNode* h);



int main()
{
    char command;
    int key;
    listNode* headnode=NULL;
    printf("-------------- ohjaesik ---- 2022040014------------ \n");
    do{
        printf("----------------------------------------------------------------
\n");
        printf("                  Doubly Circular Linked List
\n");
```

```
        printf("----------------------------------------------------------------
\n");
        printf(" Initialize    = z           Print         = p \n");
        printf(" Insert Node   = i           Delete Node   = d \n");
        printf(" Insert Last   = n           Delete Last   = e\n");
        printf(" Insert First  = f           Delete First  = t\n");
        printf(" Invert List   = r           Quit          = q\n");
        printf("----------------------------------------------------------------
\n");

        printf("Command = ");
        scanf(" %c", &command);

        switch(command) {
        case 'z': case 'Z':
            initialize(&headnode);
            break;
        case 'p': case 'P':
            printList(headnode);
            break;
        case 'i': case 'I':
            printf("Your Key = ");
            scanf("%d", &key);
            insertNode(headnode, key);
            break;
        case 'd': case 'D':
            printf("Your Key = ");
            scanf("%d", &key);
            deleteNode(headnode, key);
            break;
        case 'n': case 'N':
            printf("Your Key = ");
            scanf("%d", &key);
            insertLast(headnode, key);
            break;
        case 'e': case 'E':
            deleteLast(headnode);
            break;
        case 'f': case 'F':
            printf("Your Key = ");
            scanf("%d", &key);
            insertFirst(headnode, key);
            break;
        case 't': case 'T':
            deleteFirst(headnode);
            break;
        case 'r': case 'R':
            invertList(headnode);
            break;
        case 'q': case 'Q':
```

```
            freeList(headnode);
            break;
        default:
            printf("\n       >>>>>   Concentration!!   <<<<<     \n");
            break;
        }

    }while(command != 'q' && command != 'Q');

    return 1;
}


int initialize(listNode** h) {

    /* headNode가 NULL이 아니면, freeNode를 호출하여 할당된 메모리 모두 해제 */
    if(*h != NULL)
        freeList(*h);

    /* headNode에 대한 메모리를 할당하여 리턴 */
    *h = (listNode*)malloc(sizeof(listNode));
    (*h)->rlink = *h;
    (*h)->llink = *h;
    (*h)->key = -9999;
    return 1;
}

/* 메모리 해제 */
int freeList(listNode* h){ // 처음 노드부터 오른쪽 순회하며 이전 노드를 삭제
    listNode *p = h->rlink;
    listNode *prev =NULL;
    if(p == NULL)
        return 0;
    while(p->rlink != NULL){
        prev = p;
        p = p->rlink;
        free(prev);
    }
    free(h);
    return 0;
}



void printList(listNode* h) {
    int i = 0;
    listNode* p;

    printf("\n---PRINT\n");
```

```
    if(h == NULL) {
        printf("Nothing to print....\n");
        return;
    }

    p = h->rlink;

    while(p != NULL && p != h) {
        printf("[ [%d]=%d ] ", i, p->key);
        p = p->rlink;
        i++;
    }
    printf("  items = %d\n", i);


    /* print addresses */
    printf("\n---checking addresses of links\n");
    printf("-------------------------------\n");
    printf("head node: [llink]=%p, [head]=%p, [rlink]=%p\n", h->llink, h, h-
>rlink);

    i = 0;
    p = h->rlink;
    while(p != NULL && p != h) {
        printf("[ [%d]=%d ] [llink]=%p, [node]=%p, [rlink]=%p\n", i, p->key, p-
>llink, p, p->rlink);
        p = p->rlink;
        i++;
    }

}



/**
 * list 에 key 에 대한 노드하나를 추가
 */
int insertLast(listNode* h, int key) { //key 값을 저장할 공간을 할당하고 저장 후에 가장
연결리스트의 가장 끝을 찾아 삽입 후 삽입된 노드의 왼쪽 엣지를 이전노드와 연결, 기존의 마지막
노드의 오른쪽 엣지를 삽입된 노드와 연결한다.
    listNode * inserted = (listNode*)malloc(sizeof(listNode));
    inserted->key = key;
    inserted->llink = NULL;
    inserted->rlink = NULL;
    if(h->rlink == h){
        h->rlink = inserted;
        inserted->llink = h;
        inserted->rlink = h;
        h->llink = inserted;
        return 1;
```

```
    }
    listNode *pre = h;
    while(pre->rlink != NULL && pre->rlink != h){
        pre = pre->rlink;
    }
    pre->rlink = inserted;
    inserted->llink = pre;
    inserted->rlink = h;
    h->llink = inserted;
    return 1;
}


/**
 * list 의 마지막 노드 삭제
 */
int deleteLast(listNode* h) { // 마지막 노드까지 이동 후에 마지막 노드 이전 노드 오른쪽 노드에 h, h 에 왼쪽 노드에 마지막노드 이전 노드를 연결 후 마지막 노드 삭제
    if(h->rlink==NULL)  // h 가 가리키는 곳의 rlink 가 가리키는 곳이 NULL 이라면
    {
        printf("nothing to delete.\n"); // 삭제할 것이 없다는 문구 출력
        return 0;
    }
    listNode* p = h;
    while(p->rlink != NULL && p->rlink != h){
        p = p->rlink;
    }
    p->llink->rlink = h;
    h->llink = p->rlink;
    free(p);
    return 1;
}


/**
 * list 처음에 key 에 대한 노드하나를 추가
 */
int insertFirst(listNode* h, int key) { // 새로 삽입된 노드의 오른쪽에 가장 앞에 있던 노드 연결 앞에있던 노드의 왼쪽을 새로 삽입된 노드로 연결, 헤더의 오른쪽에 새로 삽입된 노드 연결, 새로 삽입된 노드 왼쪽 헤더 연결
    listNode *node = (listNode*)malloc(sizeof(listNode));
    node->key = key;
    node->rlink = h->rlink;
    h->rlink->llink = node;
    h->rlink = node;
    node->llink = h;

    return 1;
}
```

```c
/**
 * list 의 첫번째 노드 삭제
 */
int deleteFirst(listNode* h) { //처음 노드의 오른쪽 노드를 헤더의 오른쪽과 연결, 처음 노드의 오른쪽 노드의 왼쪽을 헤더와 연결 이후 처음 노드 삭제
    if(h->rlink == NULL)    // h 가 가리키는 곳의 rlink 가 가리키는 곳이 NULL 이라면
    {
        printf("nothing to delete.\n"); // 삭제할 것이 없다는 문구 출력
        return 0;
    }

    listNode *node = h->rlink;
    h->rlink = node->rlink;
    node->rlink->llink = h;
    free(node);

    return 1;

}


/**
 * 리스트의 링크를 역순으로 재 배치
 */
int invertList(listNode* h) { // n 이 NULL 이 아닐때 까지 trail 에 middle 의 주소를 넣고, middle 에 n 의 주소, n 에 rlink 의 주소, middle 의 rlink 에 trail 의 주소, middle 의 llink 에 n 의 주소를 넣는 반복문.
    if(h->rlink==NULL)  // h 가 가리키는 곳의 rlink 가 가리키는 곳이 NULL 이라면
    {
        printf("nothing to invert...\n");   // 역순으로 할 것이 없다는 문구 출력
        return 0;
    }
    listNode *n = h->rlink;
    listNode *middle = NULL;
    listNode *trail = NULL;
    while(n!= NULL && n != h){
        trail = middle;
        middle = n;
        n = n->rlink;
        middle->rlink = trail;
        middle->llink = n;
    }
    h->llink = h->rlink; //h 의 llink 에 h->rlink 주소값 삽입.
    h->rlink->rlink = h; // h 의 rlink 의 rlink 에 h 삽입.
    h->rlink = middle; // h 의 rlink 의 middle 삽입.
```

```c
    return 0;
}



/* 리스트를 검색하여, 입력받은 key 보다 큰값이 나오는 노드 바로 앞에 삽입 */
int insertNode(listNode* h, int key) {
    listNode* node = (listNode*)malloc(sizeof(listNode));
    node->key = key;
    node->llink = node->rlink = NULL;

    if(h->rlink==h){ //리스트의 rlink 가 비어있다면 rlink 에 삽입.
        h->rlink = node;
        node->llink = h;
        h->llink = node;
        node->rlink = h;
        return 0;
    }
    listNode* n = h->rlink;

    while(n != NULL && n != h){ //삽입받은 key 보다 큰 값을 찾는 반복문.
        if(n->key >= key){ // 삽입받은 key 보다 존재한다면 알맞은 위치에 삽입.
            if(n == h->rlink){ //n 이 rlink 와 같다면 가장 앞에 삽입.
                insertFirst(h,key);
            }
            else{ // 그렇지 않다면 node 의 rlink 에 n, llink 에 n->llink 삽입. n->llink->rlink 에 node, n->llink 에 node 삽입.
                node->rlink = n;
                node->llink = n->llink;
                n->llink->rlink = node;
                n->llink = node;
            }
            return 0;
        }
        n = n->rlink;
    }
    insertLast(h,key); //반복문이 끝나도 함수가 종료되지 않았을시 가장 뒤에 삽입.
    return 0;
}


/**
 * list 에서 key 에 대한 노드 삭제
 */
int deleteNode(listNode* h, int key) {
    if(h->rlink == h){ //h->rlink 가 h 라면 삭제할것이없다는것을 출력
        printf("nothing to delete. \n");
        return 1;
    }
```

```
    listNode* n = h->rlink;
    while(n!=NULL && n!= h){ //n을 처음 부터 끝까지 반복하는 반복문.
        if(n->key == key){ //삭제할 key 값과 노드의 key 값이 일치하면
            if(n == h->rlink){ //만약 처음 노드라면 deleteFirst
                deleteFirst(h);
            }
            else if(n->rlink == NULL){ //마지막 노드라면 deleteLast
                deleteLast(h);
            }
            else{ //그렇지 않다면 n->link->rlink 에 n->rlink 를 , n->rlink->llink 에 n->llink 를 연결 후 free
                n->llink->rlink = n->rlink;
                n->rlink->llink = n->llink;
                free(n);
            }
            return 1;
        }
        n = n->rlink;
    }
    printf("cannot find the node for key = %d\n", key); //삭제할 것이 없을 때 문구 출력
    return 0;
}
```

```
Last login: Wed May 17 19:38:22 on ttys001
➜  자료구조 git:(main) ✗ git push origin main
^[[A^[[BEnumerating objects: 121, done.
Counting objects: 100% (121/121), done.
Delta compression using up to 8 threads
Compressing objects: 100% (104/104), done.
Writing objects: 100% (121/121), 4.69 MiB | 329.00 KiB/s, done.
Total 121 (delta 42), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (42/42), done.
To https://github.com/ohjaesik/homework8.git
 * [new branch]      main -> main
➜  자료구조 git:(main) ✗ git rm --cached doubly-linked-list.c
rm 'doubly-linked-list.c'
➜  자료구조 git:(main) ✗ git commit -m '더블연결리스트 삭제'
[main cd6777b] 더블연결리스트 삭제
 Committer: 오재식 <ojaesig@ojaesigs-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 379 deletions(-)
 delete mode 100644 doubly-linked-list.c
➜  자료구조 git:(main) ✗ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 268 bytes | 268.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ohjaesik/homework8.git
   94bc3b5..cd6777b  main -> main
➜  자료구조 git:(main) ✗
```

```
Command = i
Your Key = 1
================================================================
                  Doubly Circular Linked List
----------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
----------------------------------------------------------------
Command = n
Your Key = 2
----------------------------------------------------------------
                  Doubly Circular Linked List
----------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
----------------------------------------------------------------
Command = f
Your Key = 3
----------------------------------------------------------------
                  Doubly Circular Linked List
----------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
----------------------------------------------------------------
Command = r
----------------------------------------------------------------
                  Doubly Circular Linked List
----------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
----------------------------------------------------------------
Command = p

---PRINT
[ [0]=2 ] [ [1]=1 ] [ [2]=3 ]   items = 3

---checking addresses of links
-------------------------------
head node: [llink]=0x147606b50, [head]=0x147704080, [rlink]=0x1480040a0
[ [0]=2 ] [llink]=0x147704080, [node]=0x1480040a0, [rlink]=0x148004080
[ [1]=1 ] [llink]=0x1480040a0, [node]=0x148004080, [rlink]=0x147606b50
[ [2]=3 ] [llink]=0x148004080, [node]=0x147606b50, [rlink]=0x147704080
----------------------------------------------------------------
                  Doubly Circular Linked List
```

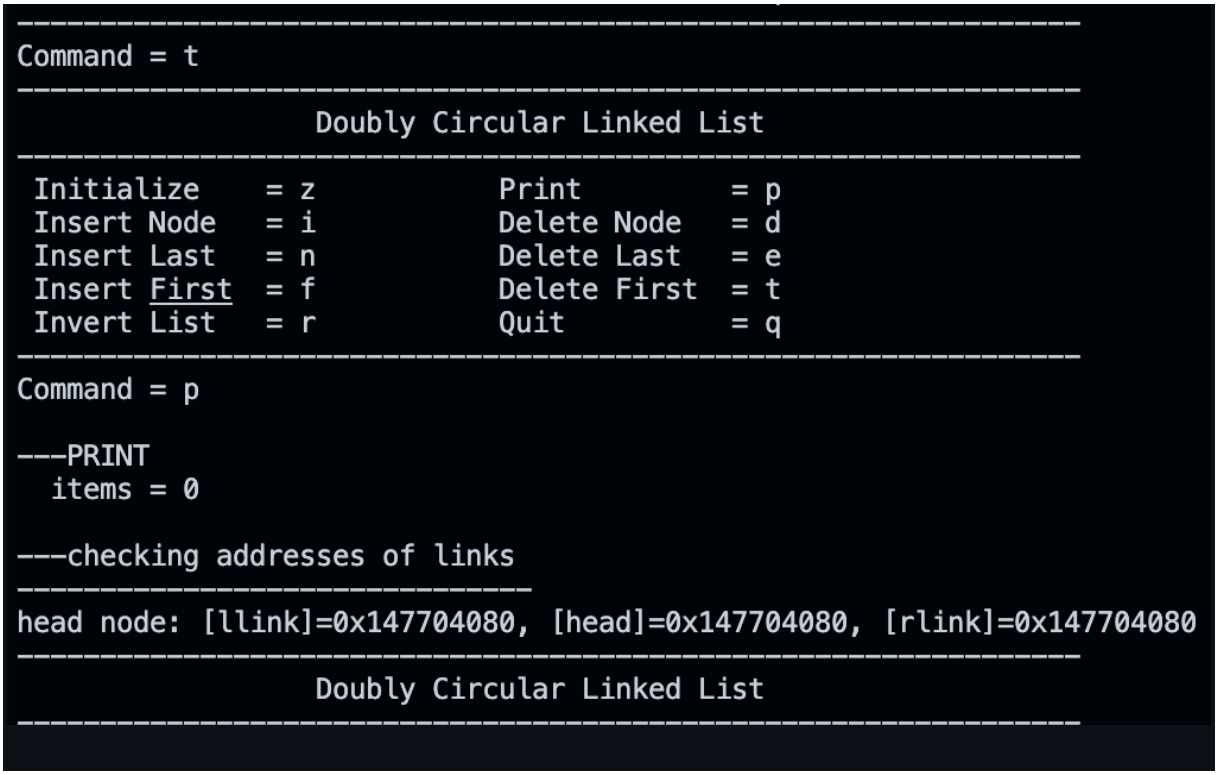
```
------------------------------------------------------------------
Command = d
Your Key = 1
------------------------------------------------------------------
                  Doubly Circular Linked List
------------------------------------------------------------------
 Initialize    = z           Print         = p
 Insert Node   = i           Delete Node   = d
 Insert Last   = n           Delete Last   = e
 Insert First  = f           Delete First  = t
 Invert List   = r           Quit          = q
------------------------------------------------------------------
Command = p

---PRINT
[ [0]=2 ] [ [1]=3 ]   items = 2

---checking addresses of links
--------------------------------
head node: [llink]=0x147606b50, [head]=0x147704080, [rlink]=0x1480040a0
[ [0]=2 ] [llink]=0x147704080, [node]=0x1480040a0, [rlink]=0x147606b50
[ [1]=3 ] [llink]=0x1480040a0, [node]=0x147606b50, [rlink]=0x147704080
------------------------------------------------------------------
                  Doubly Circular Linked List
------------------------------------------------------------------
 Initialize    = z           Print         = p
 Insert Node   = i           Delete Node   = d
 Insert Last   = n           Delete Last   = e
 Insert First  = f           Delete First  = t
 Invert List   = r           Quit          = q
------------------------------------------------------------------
Command = e
------------------------------------------------------------------
                  Doubly Circular Linked List
------------------------------------------------------------------
 Initialize    = z           Print         = p
 Insert Node   = i           Delete Node   = d
 Insert Last   = n           Delete Last   = e
 Insert First  = f           Delete First  = t
 Invert List   = r           Quit          = q
------------------------------------------------------------------
Command = p

---PRINT
[ [0]=2 ]   items = 1

---checking addresses of links
--------------------------------
head node: [llink]=0x147704080, [head]=0x147704080, [rlink]=0x1480040a0
[ [0]=2 ] [llink]=0x147704080, [node]=0x1480040a0, [rlink]=0x147704080
```



https://github.com/ohjaesik/homework8